## 注意点

动态规划……真讲不清楚，还是直接讲题吧……

### dp边界问题

哪些状态为dp边界，通过观察状态转移方程得到，一般最值下标对应状态为边界，其值再根据实际意义来设置。

### 路径打印问题

有时要求打印出最优解，可以在dp过程中用last数组记录下得到当前最优状态的前一状态，也可在求出结果后根据递推表达式，逆序求得路径，递归打印。

## 最长公共子序列（LCS）长度

给定两个序列，每个序列中元素各不相同，求解它们的最长公共子序列长度。

首先明确子序列和子串的差别，子序列不要求连续，而子串要，二者都不能改变元素在原序列中的先后顺序。

序列A、B中元素各不相同，因此记录小A中各元素的下标，对B中元素进行相应的替换，转换成一个下标序列，对此序列求最长上升子序列即可（B中的上升可一一对应成A中的上升）。

若元素可以重复，则在A中一个元素可能有多个下标，对B中元素进行替换时，下标要用逆序，这样在求最长上升子序列时，用了下标大的就不能用下标更小的，才不会求错。

最长上升子序列（LIS）就很好求了，原序列中每个元素，要么加入之前已有的一个序列中（需要比已有序列最后一个元素大），或者自成一个新的序列（找不到可以加入的序列）。使用dp[i]表示以第i个元素结尾的最长上升子序列长度，则边界dp[1]=1（假设序列下标从1开始）。令MAX=0，之后从第1到i-1个元素找起，设所找元素下标为k，只要第i个元素大于该元素，则MAX=max(MAX,dp[k]+1)，最后若MAX仍为0（找不到一个更小的元素），则dp[i]=1（自成新序列），否则dp[i]=MAX（加入旧序列）。最后所求值则为dp[序列长度]。

这么转换是因为LIS有O(Nlog(N))的解法，转换后可能能更快，实际上也可直接用O(M\*N)的算法求，dp[i][j]为A中考虑到i，B中考虑到j时的LCS长度，则:

若A[i]==B[j]，dp[i][j]=dp[i-1][j-1]+1

若A[i]!=B[j]，dp[i][j]=max(dp[i-1][j],dp[i][j-1])

注意，dp的状态别总是套以某个下标结尾，也有可能是考虑到某个下标。

## 长公共上升子序列（LCIS）长度

设dp[i][j]为考虑到A[i]、B[j]且以B[j]结尾的LCIS长度，因为要求上升，所以当前考虑的位数必须用来比较，一次相比于LCS，加上了以B[j]结尾的状态。

当A[i]!=B[j]时，dp[i][j]=dp[i-1][j]（A[i]没用，又要保证以B[j]结尾，所以A前移一位）

当A[i]==B[j]时，同LIS，需要找B[j]>B[k]且能使dp[i-1][k]最大的，再加上1。看似寻找的过程又需要O(N)的时间，但要寻找的前提是A[i]==B[j]，而i处于外层循环，所以可以在内层循环更新大于A[i]且使dp[i-1][k]最大的值，直接使用，总复杂度O(M\*N)。

## 黑客攻击（状态压缩、集合dp、子集枚举）

n（1<=n<=16）台电脑，编号0到n-1，每台电脑运行n台服务，电脑之间可能直接相连，可以给每台电脑装病毒，终止该电脑和与之直接相连的电脑上的指定服务，求最多可以完全终止多少项服务（没有一台电脑上运行着该服务）。

每台电脑可以看做电脑本身和与之直接相连的所有其他电脑共同组成的集合，那么，只要找到若干集合为一组，组内所有集合的并集为全集（所有电脑），那就能终止一项服务，所以问题转换为给定n个集合，将它们划分成最多组数，每组内集合的并都为全集。

为了高效节约地表示状态，同时方便地进行集合的并、补等操作，可以用n个二进制位来表示一种状态（状态压缩），第i位为1表示编号为i的电脑处于集合中，所有状态有2^n个，即0到(2^n)-1，当然，每种状态只是选了“主电脑”，实际上每台电脑可能都直接与其他电脑相连，于是对于一种状态，寻找所有为1的位，然后将本状态与对应集合相或，就能记录下真正的分组并集。

用f(S)表示集合S（主电脑集合）所能划分的最大组数，则若S存在子集S0，S0的实际状态就为全集，那么有f(S)=max{f(S-S0)+1}（划分出S0为一组，加上剩下集合所能划分的最大组数），集合S有2^n个，即0到(2^n)-1，每个集合划分子集方法：

for(S0=S;S0;S0=(S0-1)&S)

求S0补集方法：

S1=S0^S

## 回文串最少分割数

给定一个字符串，问最少能将它分成几个子串，使得每一个子串都是回文串。

假设已有一个最优分割，分割成了n个子串，那么在原串中除去最后几个子串，剩下的再来分割，最优分割必然还是那样，因此找到最优子结构。

dp[i]表示以下标i字符结尾，原串开头为开头的子串的最小分割数，那么从i开始向前找回文串，假设找到的回文串起始为j（可能有多个），那么dp[i]=min{dp[j-1]+1}，起始dp[0]=0，dp[1]=1。

## 放置街灯 树形dp

在无向无环图（森林）的节点（节点数为N）上放灯，可以照亮与该点相邻的所有边，尽量使灯数少，灯数相同时，尽量使被两灯同时照亮的边数多，求解灯数和分别被一盏灯、两盏灯照亮的边数。

需要优化的变量有两个，并且一个为求最小，另一个为在前一个相同的方案中取最大，为了统一为求最小值，化为求尽可能少的灯数v1和灯数相同时求尽可能少的只被一盏灯照亮的边数。

两个量化为成求一个量更方便，可设ans=M\*v1+v2，求ans最小值，M是一个比v2最大值与最小值之差还更大的数（这样只要不同方案中v1相差1，v2相差再大都还是v1起决定性作用），最后v1=ans/M，v2=ans%M。

所给的图是森林，其中每棵树可以单独决策，最后累加。表示一个图，因为图可能是稀疏的，所以使用vector数组变长表示更加节约与快捷，如：

vector<int> x[1000]

x[i]表示的是与之相邻的所有节点组成的变长数组，多组数据时，起初每个x[i]进行clear操作，之后若读入a与b相邻，因为是无向图，于是进行操作：

x[a].push\_back(b);

x[b].push\_back(a);

要决策就必然要遍历树，遍历需要有方向，而原图是无向图，实际上任意选一个点作为根，进行dfs就行了，决策也可在dfs的过程中进行，要遍历就必然要标记已经走过的点防止重复走，使用一个bool型数组vis[1000][2]即可，数组有二维且第二维大小为2，是因为对每个点有2中决策，因此与其说遍历树，不如说遍历树上的决策。

父节点上放与不放会影响到子节点的决策，因此状态中应加入父节点是否以放这一状态，所以上面说每个点有两种决策。dp[1000][2]表示第i个点的父节点放灯状态为j（1为放，0为未放）时的最小决策值，每个点有两种决策：

放（无条件的，无论如何都可以放）：ans1+=M，若存在父节点且父节点未放ans1还需加1，然后ans1再加上所有子树的相应决策。

不放（有条件的），只有当本节点为根节点，或者父节点放了（此时ans2++）才可以不放，然后ans2再加上所有子树的相应决策。之后在这一条件中使ans1取ans1与ans2中的最小值。

最后，dp[i][j]=ans1。

## 捡垃圾的机器人

有n个从小到大编好了号的垃圾，都于第一象限，第i号垃圾坐标为(x,y)，且有重量wi，一个机器人每次只能从原点起按次序捡起编号连续的几个垃圾，再返回原点扔掉，机器人最大载重为C，两点之间走的距离为曼哈顿距离

abs(x2-x1)+abs(y2-y1)

求机器人行走的总的最短距离，起初在原点。

假设最优决策中，机器人每次捡的垃圾将所有垃圾分成了几段，那么假设出去第i段之后的段，为第i段以及前面的垃圾决策，最终分段还会是那样，于是找到最优子结构。使用dp[i]表示将前i个垃圾全部清理完时的最短距离，求解dp[i]，则从i向前不断找j，控制j不越界以及j到i的垃圾总重量未超过C，对于每个j，记录dp[j-1]+原点到j距离+j到i距离，找使之取最小的j，即可求解dp[i]。其中涉及到许多转换和优化，见代码。

## 切巧克力 集合dp，状态压缩

World Final题？吓坏了，菜鸡当然是看了题解才会写的。

给定长宽分别为x和y的巧克力（分好了块的，想象成网格），每次只能横着或竖着切且每次只能切一块巧克力，问能否经过若干次操作将之切成n（1<=n<=15）块面积分别为a1,a2……an的巧克力。

需要切成的面积集合较小，可以通过状态压缩用二进制数表示，第i位为1代表面积area[i]处于该集合中。问题可以转换为长为j，宽为i的巧克力能否分割成面积集合S0，实际上若每次限定好S0的总面积为i\*j，可以把j略去，用dp[i][S0]表示短边为i的巧克力能否分割成面积集合S0。则为求dp[min(x,y)][S]。于是不断枚举子集了，每种子集确保总面积为短边或长边的倍数，分别代表平行于短边切和平行于长边切，只要有一种子集，一种切法可行，dp[i][S0]就为true。若S中只有1个位为1，则必然为true（面积之前已经限定好了的，不会有错）。

## 序列修改

给定一个无向连通图，点数为n1（1<=1<=100），再给定一个点序列p（元素值在点的编号范围内），长度为n，每次可对序列中一个元素进行修改，求最少修改数，使得相邻元素要么相同，要么在图中相邻。

注意到除第一个点外，每个点是否满足条件只和前面那个点的值有关，设dp[i][j]表示第i个点值改为j，且前i个点满足条件时的最少修改数，则dp[1][j]=(j==p[1]?0:1)，i大于2时，若j==p[i]（该点无需修改），dp[i][j]=min(dp[i][j], dp[i-1][k])，其中k是与j相邻的点，j!==p[i]时（该点需要修改），还应加上1。

## 最大m段子段和

给定一个数组，可任意选取其不相交的m段（不必选完整个数组），问这m段和的最大值是多少。

待理解。